Solving optimization problems simultaneously: the variants of the traveling salesman problem with time windows using multifactorial evolutionary algorithm

Ha-Bang Ban [Corresp., 1], **Dang-Hai Pham** [1]

[1] Computer Science, Hanoi University of Science and Technology, Hanoi, Vietnam

Corresponding Author: Ha-Bang Ban
Email address: BANGBH@SOICT.HUST.EDU.VN

We study two problems called the Traveling Repairman Problem (TRPTW) and Traveling Salesman Problem (TSPTW) with time windows. The TRPTW wants to minimize the sum of travel durations between a depot and customer locations, while the TSPTW aims to minimize the total time to visit all customers. In two problems, the deliveries are made during a specific time window given by the customers. The difference between the TRPTW and TSPTW is that the TRPTW takes a customer-oriented view, whereas the TSPTW is server-oriented. Existing algorithms have been developed for solving two problems independently in the literature. However, an algorithm that simultaneously solves two problems does not exist in the literature. Nowadays, Multifactorial Evolutionary Algorithm (MFEA) has been a variant of the Evolutionary Algorithm (EA), aiming to solve multiple factorial tasks simultaneously. The main advantage of the approach is to allow transferrable knowledge between tasks. Therefore, it can improve the solution quality for multitasks. This paper presents an efficient algorithm that combines the MFEA framework and Randomized Variable Neighborhood Search (RVNS) to solve two problems simultaneously. The proposed algorithm has transferrable knowledge between tasks from the MFEA and the ability to exploit good solution space from RVNS. Experiment results show the impressive efficiency of the proposed MFEA.

Solving Optimization Problems Simultaneously: The variants of the Traveling Salesman Problem with Time Windows using Multifactorial Evolutionary Algorithm

Ha-Bang Ban, Dang-Hai Pham

School of Information and Communication Technology, Hanoi University of Science and Technology, Hanoi, Vietnam

Corresponding author:
Ha-Bang Ban

Email address: BangBH@soict.hust.edu.vn

ABSTRACT

We study two problems called the Traveling Repairman Problem (TRPTW) and Traveling Salesman Problem (TSPTW) with time windows. The TRPTW wants to minimize the sum of travel durations between a depot and customer locations, while the TSPTW aims to minimize the total time to visit all customers. In two problems, the deliveries are made during a specific time window given by the customers. The difference between the TRPTW and TSPTW is that the TRPTW takes a customer-oriented view, whereas the TSPTW is server-oriented. Existing algorithms have been developed for solving two problems independently in the literature. However, an algorithm that simultaneously solves two problems does not exist in the literature. Nowadays, Multifactorial Evolutionary Algorithm (MFEA) has been a variant of the Evolutionary Algorithm (EA), aiming to solve multiple factorial tasks simultaneously. The main advantage of the approach is to allow transferrable knowledge between tasks. Therefore, it can improve the solution quality for multitasks. This paper presents an efficient algorithm that combines the MFEA framework and Randomized Variable Neighborhood Search (RVNS) to solve two problems simultaneously. The proposed algorithm has transferrable knowledge between tasks from the MFEA and the ability to exploit good solution space from RVNS. Experiment results show the impressive efficiency of the proposed MFEA.

1 INTRODUCTION

Evolutionary Algorithms based on natural evolutionary inspiration are popular optimization techniques for various fields. Recently, a new approach called the MFEA (K.K. Bali et al.,2019; A. Gupta et al.,2016; E. Osaba et al.,2020; Y. Yuan et al.,2016; Q. Xu et al.,2021) has been proposed to incorporate the characteristics of Evolutionary Algorithms into multitasking to handle multiple search spaces of multiple problems simultaneously. The main advantage is that the phenomenon of implicit genetic transfer in multitasking exploits transferrable knowledge between optimization tasks, thereby facilitating improved performance characteristics for multiple tasks simultaneously. It also can be useful in a system with limited computation.

The TSPTW (W.B. Carlton et al.,1996; S. Dash et al.,2012; M. Dorigo et al.,1997; Y. Dumas et al.,1995; M. Gendreau et al.,1998; A. Langevin et al.,1993; R. F. Silva et al.,2010; J. N. Tsitsiklis et al.,1992; J. W. Ohlmann et a G. Pesant et al.,1998; R. Roberti et al.,2016), and TRPTW (H.Abeledo et al.,2013; H.B. Ban et al.,2017; H.B. Ban et al.,2021; G. Heilporna et al.,2010; J. N. Tsitsiklis et al.,1992) are combinatorial optimization problems that have many practical situations. The TRPTW wants to minimize the sum of travel durations between a depot and customer locations, while the TSPTW aims to minimize the total time to visit all customers. In two problems, the deliveries are made during a specific time window given by the

customers. We can formulate two problems as follows:
We have an complete graph $K_n = (V, E)$, where $V = v_1, v_2, ..., v_n$ is a set of vertices showing the starting vertex and customer locations, and E the set of edges connecting the customer locations. Suppose that, in a tour $T = (v_1 = s, v_2 ..., v_n)$, each edge $(v_i, v_j) \in E$ connecting the two vertices v_i and v_j there exists a cost $c(v_i, v_j)$. This cost represents the travel time between vertex v_i and v_j. Each vertex $v_i \in V$ has a time window $[e_i, l_i]$ indicating when starting service time at vertex v_i. This implies that a vertex v_i may be reached before the start e_i, but service cannot start until e_i and no latter than l_i of its time window. Moreover, to serve each customer, the salesman spends a mount of time. Let $D(v_i), S(v_i)$ be the time at which service begins and the service time at vertex v_i. It is calculated as follows: $D(v_i) = \max\{A(v_i), e_i\}$, where $A(v_k) = D(v_{i-1}) + S(v_{i-1}) + c(v_{i-1}, v_i)$ is the arrival time at vertex v_i in the tour. A tour is feasible, if and only if $A(v_i) \leq l_i$ for all vertices. The objective functions of two problems is defined as follows:

- In the TSPTW, the salesman must return to s. Therefore, the cost of the tour T is defined as: $\sum_{i=1}^{n} c(v_i, v_{i+1})$. Note that: $v_{n+1} \equiv s$.
- In the TRPTW, we also define the travel duration of vertex v_i as the difference between the beginning of service at vertex v_i and the beginning of service at s: $t_i = D(v_i) - D(s)$. The cost of the tour T is defined: $\sum_{i=2}^{n} t_i$.

Two problems consist of determining a tour, starting at the depot s, to minimize the cost of the tour overall vertices while respecting time windows.
Currently, various algorithms have been proposed to solve them. However, they solve each problem independently. This paper proposes the algorithm based on the MFEA framework to solve two problems simultaneously. The major contributions of this work are as follows:

- From the algorithmic aspect, we develop a first metaheuristic inspired by the MFEA framework. The proposed metaheuristic utilizes the advantages of the MFEA and RVNS. The MFEA with the RVNS to have positive transferrable knowledge between tasks from the MFEA and the ability to exploit good solution spaces from RVNS.
- From the computational aspect, numerical experiments show that the proposed algorithm reaches nearly optimal solutions in a short time for two problems simultaneously. Moreover, it obtains better solutions than the previous MFEA algorithms in many cases.

The rest of this paper is organized as follows. Sections 2 and 3 present the literature and preliminary, respectively. Section 4 describes the proposed algorithm. Computational evaluations are reported in Section 5. Section 6 discusses and concludes the paper, respectively.

2 LITERATURE AND OUR METHODOLOGY

2.1 The MFEA literature

The MFEA (K.K. Bali et al.,2019; A. Gupta et al.,2016; E. Osaba et al.,2020; Y. Yuan et al.,2016; Q. Xu et al.,2021) have been known as an efficient framework for solving many optimization problems. The main advantage of the MFEA framework is to solve multiple well problems simultaneously. The advantage is that the genetic transfer occurs in relevant multitasking tasks to facilitate finding optimal solutions for multiple problems simultaneously.

Recently, several variants of the MFEA framework are also introduced. Y. Yuan (Q. Xu et al.,2021) developed evolutionary multitasking in permutation-based optimization problems. They tested it on several popular combinatorial problems. The experiment results indicated the promising scalability of evolutionary multitasking to many-task environments. E. Osaba et al. (E. Osaba et al.,2020) then proposed a dMFEA-II framework to exploit the complementarities among several tasks, which was often achieved via genetic information transfer. The experiments on some combinatorial optimization problems showed the impressive performance of the dMFEA-II. This paper considers these works as a baseline in our research.

2.2 The TSPTW and TRPTW literature

The TSPTW and TRPTW have many practical applications in scheduling tasks and logistics, e.g., In the TRPTW, it aims at finding a tour to minimize the sum of travel durations between a depot and all

customers in the TRPTW while the sum of tour length is minimized in the TSPTW. Each customer must be visited within a given time window in two problems. Due to time window constraints, the TSPTW and TRPTW are much harder than the traditional TSP and TRP.

The Travelling Salesman Problem with Time Windows (TRPTW) is a popular NP-hard combinatorial optimization problem studied much in the literature (Y. Dumas et al.,1995). The algorithms include exact and metaheuristic approaches. Langevin et al. (A. Langevin et al.,1993) introduced a two-commodity flow formulation to solve the problem. Dumas et al. (Y. Dumas et al.,1995) used a dynamic programming approach while Pesant et al. (G. Pesant et al.,1998) proposed a branch-and-bound algorithm using a constraint programming model. Similarly, Focacci et al. (M. Gendreau et al.,1998) brought constraint programming and optimization techniques together. Most recently, Dash et al. (S. Dash et al.,2012) propose a method using an IP model based on discretization of time. The results are extremely good: several benchmark instances are solved first.

M. Dorigo et al. (M. Dorigo et al.,1997) proposed an algorithm based on Ant Colony Algorithm to solve the problem. Carlton et al. (W.B. Carlton et al.,1996) then combined a tabu-search with a penalty function. Their algorithm accepted infeasible solutions. Gendreau et al. (M. Gendreau et al.,1998) proposed an insertion heuristic that generated the solution in the first step and improved it in a post-phase using removal and reinsertion of vertices. Ohlmann et al. (J. W. Ohlmann et al.,2007) developed simulated annealing relaxing the time windows constraints by integrating a variable penalty method within a stochastic search procedure. In this work, we developed a two-phase heuristic. In the first phase, a feasible solution was created by using a Variable Neighborhood Search (VNS). In the post phase, this solution was improved by using a General VNS. R. Roberti et al. (R. Roberti et al.,2016) proposed a mathematical model for solving optimality instances with small sizes. In addition, good solutions were obtained from the heuristic for the instances with large sizes. Generally speaking, the results from this approach are very promising.

In the Travelling Repairman Problem with Time Windows (TRPTW), there are an exact algorithm and three metaheuristics algorithms in the literature: 1) Tsitsiklis (J. N. Tsitsiklis et al.,1992) proposed a polynomial algorithm when several customers are bounded; 2) Heilporn et al. (G. Heilporna et al.,2010) then developed an exact algorithm and heuristic algorithm to solve the problem; 3) Ban et al. (H.B. Ban et al.,2017; H.B. Ban e proposed two metaheuristic algorithms based on Variable Neighborhood Search (VNS) scheme. Their experimental results showed the efficiency of the metaheuristic approach.

We consider an example that describes the difference between two problems in a specific instance. If we use the optimal solution of n40w160.002 instance for the TSPTW [1], the objective function cost (using the function cost of the TRPTW) of this solution for the TRPTW is 7519, while the known-best cost for this instance for the TRPTW is 6351 (the known-best cost is found by our algorithm). Thus, the difference between the two objective function costs is 15.5%. It implies that a good metaheuristic algorithm for the TSPTW does not produce a good solution for the TRPTW and vice versa. The above algorithms are the best algorithms for two problems. However, they only solve each problem independently, but they cannot simultaneously produce good solutions for two problems.

2.3 Our methodology

For NP-hard problems, we have three common approaches to solve the problem, specifically, 1) exact algorithms, 2) approximation algorithms, 3) heuristic (or metaheuristic) algorithms:

- The exact algorithms guarantee to find the optimal solution. However, they are exponential time algorithms in the worst case.
- An α-approximation algorithm produces a solution within some factor of α of the optimal solution.
- Heuristic (metaheuristic) algorithms perform well in practice and validate their performance through experiments. This approach is suitable for a problem with large sizes.

Previously, several metaheuristics have been proposed to solve the TSPTW (W.B. Carlton et al.,1996; M. Dorigo et al.,19 and the TRPTW (H.B. Ban et al.,2017; H.B. Ban et al.,2021; G. Heilporna et al.,2010; J. N. Tsitsiklis et al.,1992). However, they are developed to solve each problem independently and separately. Therefore, they cannot solve both two problems well simultaneously. When we run the two best algorithms for two tasks independently, there is no transferrable knowledge between tasks, and we cannot improve solution quality.

[1] https://homepages.dcc.ufmg.br/ rfsilva/tsptw/

This paper proposes a new approach using the MFEA framework to solve two problems simultaneously. Our MFEA has two tasks simultaneously: the first is solving the TRPTW, and the second is solving the TSPTW. Experiment results show its efficiency: 1) for small cases, the algorithm reaches the optimal solutions in both of two problems; 2) for large cases, our solutions are nearly optimal solutions, even much better than those of the previous MFEA approaches.

3 PRELIMINARY

The concept of multifactorial optimization (MFO) is introduced in (K.K. Bali et al.,2019; A. Gupta et al.,2016). After that, we describe briefly how to incorporate the concept of MFO in EA. Assume that, k optimization problems are needed to be performed simultaneously. Without loss of generality, all tasks are assumed to be minimization problems. The j-th task, denoted T_i, has objective function $f_j : X_j \Rightarrow R$, in which x_j is solution space. We need to be found k solutions $\{x_1, x_2, ..., x_{k-1}, x_k\} = \min\{f_1(x), f_2(x), ..., f_{k-1}(x), f_k(x)\}$, where x_j is a feasible solution in X_j. Each f_j is considered as an additional factor that impacts the evolution of a single population of individuals.Therefore, the problem also is called $k-$ factorial problem. For a composite problem, general method for comparing individuals is necessary. Each individual $p_i (i \in \{1, 2, ..., |P|\})$ in a population P has a set of properties as follows: Factorial Cost, Factorial Rank, Scalar Fitness, and Skill Factor. These properties allow us to sort, and select individuals in the population.

- Factorial Cost c_j^i of the individual p_i is its fitness value for task T_j $(1 \leq j \leq k)$.
- Factorial rank r_j^i of p_i on the task T_j is its index in the list of population individuals sorted in ascending order with respect to c_j^i.
- Scalar-fitness ϕ_i of p_i is given by its best factorial rank overall tasks as $\phi_i = \frac{1}{\min_{j \in 1,...,k} r_j^i}$.
- Skill-factor ρ_i of p_i is the one task, amongst all other tasks, on which the individual is most effective, i.e., $\rho_i = argmin_j\{r_j^i\}$ where $j \in \{1, 2, ..., k\}$.

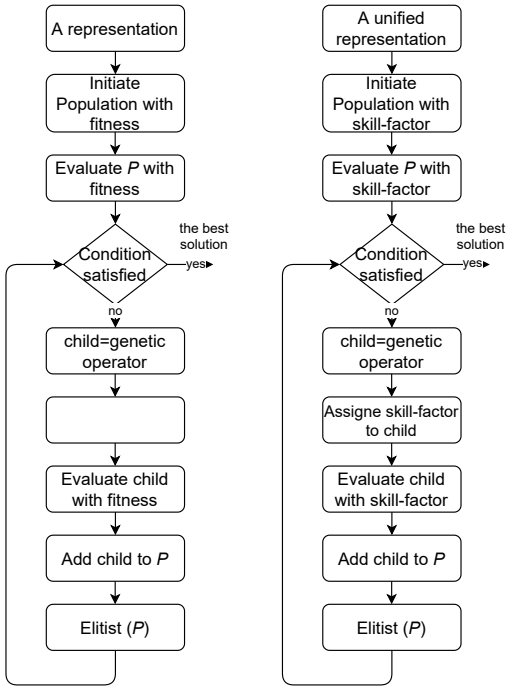
The pseudo-code of the basic MFEA is described in Figure 1: We first build the unified search space that encompasses all individual search spaces of different tasks to have a shared background on which the transfer of information can take place. We then initialize *SP* individuals (*SP* is the size of population) in the unified search space and then evaluate it by calculating the skill-factor of each individual. After the initialization, the iteration begins to produce the offsprings and assign skill-factors to them. Selective evaluation guarantees that the skill-factor of each new offspring is selected randomly among those of the parents. The offspring and the parent are merged in a new population with $2 \times SP$ individuals. The evaluation for each individual is taken only on the assigned task (in the step, the best solution for each task is updated if it is found. This best solution for each task is the output). After evaluation, the individuals of the population receive new skill-factors. The Elitist strategy keeps the *SP* solutions with the best skill-factors for the next generation.

Figure 1 also shows the differences between the traditional EA and MFEA. The crossover and mutation operators in the MFEA are like the traditional EA. However, there are two different important aspects: 1) the parents' skill-factor and 2) random mating probability (*rmp*). Specifically, the child is created using crossover from parents with the same skill-factor. Otherwise, the child is generated by a crossover with a *rmp* value or by a mutation when parents own different skill-factors. A large *rmp* value generates more information exchanging between tasks. Also, in the traditional GA, the fitness of child is evaluated directly while the skill-factor is assigned to it in the MFEA.

The MFEA is also different from multi-objective optimization. In multi-objective optimization, only one problem with many objective functions is solved, while the MFEA aims to optimize many tasks simultaneously. In addition, multi-objective optimization generally uses a single representative space, while the MFEA unifies multiple representative spaces for many tasks.

Running two algorithms for two tasks independently is not the idea of MFEA approach. When two algorithms run independently, each task is represented by its own search space. There is no transferrable knowledge between tasks. Otherwise, in the MFEA, two tasks use the unified search space, and transferrable knowledge between tasks is done. It can increase convergence and improve the quality of solutions for multitasks. Lian et al. (Y. C. Lian et al.,2019) then provided a novel theoretical analysis and evidence of the efficiency of the MFEA. This study theoretically explains why some existing the MFEA

Figure 1. The similarity and difference between EA and MFEA



perform better than traditional EAs. In addition, the MFEA also can be useful in a system with limited computation.

4 THE PROPOSED ALGORITHM

This section introduces the pseudocode of the proposed MFEA. The TSPTW task corresponds to a particular task in the MFEA, while another is the TRPTW task. The proposed algorithm is shown in Algorithm 1. Our MFEA has five core components: unified representation, assortative mating (crossover and mutation operators), selective evaluation, scalar-fitness-based selection, and RVNS. More specifically, the algorithm includes the following steps. In the first step, a unified search space is created for both two problems. The population with *SP* individuals is then generated in the second step. All solutions in the population must be feasible. After that, the iteration begins until the termination criterion is satisfied. Parents are selected to produce offsprings using crossover or mutation and then assign skill-factors to them. The offsprings then are added to the current population. The individuals of the population are evaluated to update their scalar-fitness and skill-factor. We select the best solution in terms of skill-factor from the current population and convert it from the unified representation to each task's one. It then is fed into the RVNS step to find the best solution for each task. The output of the RVNS is then converted to the unified search space. If an improvement is found, it will replace the old solution in the population. The Elitist strategy keeps the *SP* solutions with the best skill-factors for the next generation.

4.1 Creating Unified Search Space-USS

For two problems, many representations are proposed in the literature. These works indicate the efficiency of permutation representation in comparison with the others. In this paper, the permutation encoding is used, in which an individual is represented as a list of n vertices $(v_1, v_2, ..., v_k, ..., v_n)$, where v_k is the $k-$th vertex to be visited. Figure 2 depicts this encoding for two tasks.

4.2 Initializing population

Each feasible solution is created from the RVNS to take a role as an individual in the population. Therefore, we have Sp individuals in the initial population for the genetic step.

Algorithm 2 describes the constructive step. The objective function is the sum of all positive differences between the arrival time $(D(v_i))$ and its due time (l_i) on each vertex. Specifically, it is

Algorithm 1 MFEA-RVNS

Require: K_n, C_{ij}, v_1, SP are the graph, the cost matrix, the starting vertex, and the size of population.
Ensure: The best solution T^*_{TSPTW}, T^*_{TRPTW}.

$T^*_{TSPTW}, T^*_{TRPTW} \longrightarrow Inf$; {Initiate the best solution for the TSPTW, TRPTW}
$P =$ Construction$(v_1, V, k, \alpha, level)$; {Initiate the population}
while (The termination criterion of the MFEA is not satisfied) **do**
  for $(j = 1; j \leqslant SP; j++)$ **do**
    (P, M) = Rank-Selection(P, NG);
    if M and P have the same skill-factor or (rand(1) $\leqslant rmp$) **then**
      C_1, C_2 = Crossover(P, M);
      if (C_1 or C_2 is infeasible) **then**
        if C_1 is infeasible **then**
          $C_1 =$ Repair(C_1);
        end if
        if C_2 is infeasible **then**
          $C_2 =$ Repair(C_2);
        end if
      end if
      C_1, C_2's skill-factors are set to P or M randomly;
    else
      C_1 = Mutate(P); C_2 = Mutate(M);
      if (C_1 or C_2 is infeasible) **then**
        if C_1 is infeasible **then**
          $C_1 =$ Repair(C_1);
        end if
        if C_2 is infeasible **then**
          $C_2 =$ Repair(C_2);
        end if
      end if
      C_1's, C_2's skill-factor is set to P, M, respectively;
    end if
    $P = P \cup \{C_1, C_2\}$;
  end for
  Update scalar-fitness and skill-factor for all individuals in P;
  T = Select the best individual from P;
  {RVNS step}
  (T_{TSPTW}, T_{TRPTW}) = Convert T from unified representation to one for each task;
  $T^{'}_{TSPTW}$ = RVNS(T_{TSPTW});
  if ($T^{'}_{TSPTW} < T^*_{TSPTW}$) **then**
    $T^{'}_{TSPTW} \longrightarrow T^*_{TSPTW}$;
  end if
  $T^{'}_{TRPTW}$ = RVNS(T_{TRPTW});
  if ($T^{'}_{TRPTW} < T^*_{TRPTW}$) **then**
    $T^{'}_{TRPTW} \longrightarrow T^*_{TRPTW}$;
  end if
  $T^{'}$ = convert$(T^{'}_{TSPTW}, T^{'}_{TRPTW})$ to unified representation;
  if ($T^{'}$ is improved) **then**
    Replace T by $T^{'}$ in P;
  end if
  P = Elitism-Selection(P);
end while
return T^*_{TSPTW}, T^*_{TRPTW};

Algorithm 2 Construction

Require: $v_1, V, k, \alpha, level$ are a starting vertex, the set of vertices in K_n, the number of vehicles and the size of RCL, the parameter to control the strength of the perturbation procedure, respectively.
Ensure: An initial solution T.

$P = \phi$
while $(|P| < SP)$ **do**
  $T = \phi$; $\{T$ is a tour$\}$
  while $|T| < n$ **do**
    Create RCL that includes α nearest vertices to v_e in V; $\{v_e$ is the last vertex in $T\}$
    Select randomly vertex $v = \{v_i | v_i \in RCL$ and $v_i \notin T\}$;
    $T \leftarrow T \cup \{v_i\}$;
  end while
  if T is infeasible solution **then**
    {Convert infeasible solution to feasible one}
    T = Repair$(T, level_max, N_i(i = 1, ..., 7))$;
  end if
  $P = P \cup \{T\}$;
end while
return P;

Algorithm 3 Repair$(T, level\_max, N_i(i = 1, ..., 7))$

Require: $T, level\_max, N_i(i = 1, ..., 7)$ are the infeasible solution, the parameter to control the strength of the perturbation procedure, and the number of neighbourhood respectively.
Ensure: An feasible solution T.

$level$ = 1;
while ((T is infeasible solution) and $(level \leq level\_max)$) **do**
  T'= Perturbation$(T, level)$;
  for $i : 1 \rightarrow 6$ **do**
    $T'' \leftarrow \arg\min N_i(T')$; {local search}
    if $(L(T'' < L(T'))$ **then**
      $T' \leftarrow T''$
      $i \leftarrow 1$
    else
      $i++$
    end if
  end for
  if $L(T') < L(T)$ **then**
    $T \leftarrow T'$;
  end if
  if $L(T') == L(T)$ **then**
    $level \leftarrow 1$;
  else
    $level++$;
  end if
end while
return T;

$\min \sum_{i=1}^{n} \max(0, D(v_i) - l_i)$. The algorithm runs until it finds a feasible solution. Restricted Candidate List (RCL) is created by ordering all non-selected vertices based on a greedy function that evaluates the benefit of including them in the solution. One vertex is then chosen from RCL randomly. Since all vertices are visited, we receive a solution. If this solution is a feasible one, it is an initial solution, and this step stops. Conversely, a repair procedure based on the RVNS with many neighborhoods (D. S. Johnson et al.,1995) is invoked, and the procedure iterates until a feasible solution is reached. The solution is shaken to escape

Algorithm 4 Selection Operator(P, NG)

Require: P,NG are the population and the size of group, respectively.
Ensure: Parents C_1,C_2.
1: Select randomly the NG individuals in the P;
2: Sort them in terms of their R values;
3: C_1,C_2 = Select two individuals with the best R values;
4: return C_1,C_2;

Algorithm 5 Crossover(P, M)

Require: P,M are the parent tours, respectively.
Ensure: A new child T.
1: *type* = rand(3);
2: *rnd* = rand(2);
3: **if** (*type*==1) **then**
4: {the first type crossover is selected}
5: **if** (*rnd*==1) **then**
6: C = **PMX**(P, M);{PMX is chosen}
7: **else if** (*rnd*==2) **then**
8: C = **CX**(P, M);{CX is selected}
9: **end if**
10: **else if** (*type*==2) **then**
11: {the second type is selected}
12: **if** (*rnd*==1) **then**
13: C = **EXX**(P, M);{EXX is selected}
14: **else if** (*rnd*==2) **then**
15: C = **EAX**(P, M);{EAX is selected}
16: **end if**
17: **else if** (*type*==3) **then**
18: {type 3 is selected}
19: **if** (*rnd*==1) **then**
20: C = **SC**(P, M);{SC is selected}
21: **else if** (*rnd*==2) **then**
22: C = **MC**(P, M);{MC is selected}
23: **end if**
24: **end if**

Algorithm 6 Mutate(C)

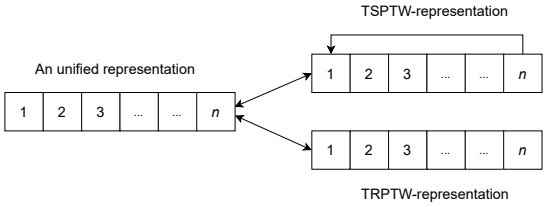
Require: C is the child tour, respectively.
Ensure: A new child C.
1: {Choose a mutation operator randomly}
2: *rnd* = rand(2);
3: **if** (*rnd*==1) **then**
4: C = **Inversion**(C){Inversion mutation is selected}
5: **else if** (*rnd*==2) **then**
6: C = **Insertion**(C){Insertion mutation is selected}
7: **else**
8: C = **Swap**(C){Swap mutation is selected}
9: **end if**
10: return C;

from the current local optimal solution. The RVNS is then applied to create the new solution. If it is better than the best-found solution, it is set to the new current solution. The *level* is increased by one if the current solution is not improved, or reset to 1, otherwise. The repair procedure is described in Algorithm

Algorithm 7 RVNS(T)

Require: T is a tour.
Ensure: A new solution T.
1: Initialize the Neighborhood List NL;
2: **while** $NL \neq 0$ **do**
3: Choose a neighborhood N in NL at random
4: $T^{'} \leftarrow \arg\min N(T)$; {Neighborhood search}
5: **if** (($W(T^{'}) < W(T)$) and ($T^{'}$ is feasible)) **then**
6: $T \leftarrow T^{'}$
7: Update NL;
8: **else**
9: Remove N from the NL;
10: **end if**
11: **if** (($W(T^{'}) < W(T^{*})$) and ($T^{'}$ is feasible)) **then**
12: $T^{*} \leftarrow T^{'}$;
13: **end if**
14: **end while**

Figure 2. The interpretation of unified representation for each task



3.

In this paper, some neighborhoods widely applied in the literature (D. S. Johnson et al.,1995). We describe more details about seven neighborhoods as follows:

- **move** moves a vertex forward one position in T.
- **shift** relocates a vertex to another position in T. .
- **swap-adjacent** attempts to swap each pair of adjacent vertices in the tour.
- **exchange** tries to swap the positions of each pair of vertices in the tour.
- **2-opt** removes each pair of edges from the tour and reconnects them.
- **Or-opt**: Three adjacent vertices are reallocated to another position of the tour.

4.3 Evaluating for individuals

The fitness function represents the method for the evaluation of individuals. First, we calculate skill-factor and scalar-fitness for each individual. The larger the scalar-fitness is, the better the individual is.

4.4 Selection operator

We propose a new selection operator for the MFEA algorithm that balances scalar-fitness and population diversity. The scalar-fitness is effectively transferred elite genes between tasks, while diversity is important when it can make a bottleneck against the genetic information transfer. For each solution, we count its scalar-fitness and its diversity in a set of solutions as follows:

$$R(T) = (SP - RF(T) + 1) + \alpha \times (SP - RD(T) + 1) \quad (1)$$

where SP, $\alpha \in [0,1]$, $RF(T)$, and $RD(T)$ are the population size, threshold, the rank of T in the P based on the scalar-fitness, and the rank of T in the P based on its diversity, respectively.

$$\overline{d}(T) = \frac{\sum_{i=1}^{n} d(T,T_i)}{n} \tag{2}$$

$d(T,T_i)$ is the distance between T, and T_i, and $\overline{d(T)}$ is the average distance of T in the population. The larger $\overline{d}(T)$ is, the higher its rank is. The larger R is, the better solution T is.

The selection operator selects individual parents based on their R values to mate. We choose the tournament selection operator (E. G. Talbi et al.,2009) because of its efficiency. A group of NG individuals is selected randomly from the population. Then, two individuals with the best R values in the group are chosen to become parents. Increased selection pressure can be provided by simply increasing the size of the group, as the winners from a larger size will, on average, have higher scalar-fitness than the winners of a small size. In this operator, selection pressure can be increased by simply increasing the size of the group, as the winners from a larger size will, on average, have higher scalar-fitness than the winners of a small size. The detail in this step is described in Algorithm 4.

4.5 Crossover operator

The crossover is implemented with the predefined probability (rmp) or if the parents have the same skill-factor. In (A. Otman et al.,2015), the crossovers are divided into three main types. We found no logical investigation showing which operator brings the best performance in the literature. In a pilot study, we found that the algorithm's performance is relatively insensitive to crossover operators, which are selected. As testing our algorithm on all operators would have been computationally too expensive, we implemented our numerical analysis on some randomly selected operators for each type. The following operators are selected from the study to balance solution quality and running time.

- The first type is related to the position of certain genes in parents (PMX, CX).
- The second selects genes alternately from both parents, without genes' repetition (EXX, EAX).
- The third is an order-based crossover (SC, MC).

Using multiple crossovers helps the population to be more diverse than using one crossover. Therefore, it can help the algorithm to prevent premature convergence. If the offsprings are infeasible, the repair procedure is invoked to convert them to feasible ones. The offspring's skill-factor is set to the one of father or mother randomly. The detail in this step is described in Algorithm 5.

4.6 Mutation operator

A mutation is used to keep the diversity of the population. Several mutations are used in our algorithm:

- The Inversion Mutation picks two vertices at random and then inverts the list of vertices between them. It preserves most adjacency information and only breaks two links, leading to the disruption of order information.
- Another simple method is the Insertion Mutation, which removes the customer from its place and inserts it in a random place on tour in another position. This preserves most of the order and the adjacency information.
- Swap Mutation selects two vertices at random and swaps their positions.

It preserves most of the adjacency information, but links broken disrupt order more. Specific time this mutation is performed, we randomly select one of three operators. After the mutation operator, two offsprings are created from the parents. If the offsprings are infeasible, the repair procedure converts them to feasible ones. Their skill-factors are set to those of parents. The detail in this step is given in Algorithm 6.

Table 1. The variable parameters

Parameter	Value Range
SP	$50 \leq \beta_r \leq 200$, incremented by 50
NG	$5 \leq \alpha \leq 15$, incremented by 5
rmp	$0.5 \leq \beta_\eta \leq 1$, incremented by 0.1
α	$5 \leq \tau_0 \leq 20$, incremented by 5
$level$	$5 \leq p \leq 15$, incremented by 5
Ng	$50 \leq Ng \leq 150$, incremeted by 50

4.7 RVNS

The combination between the MFEA with the RVNS to have good transferrable knowledge between tasks from the MFEA and the ability to exploit good solution spaces from RVNS. In the RVNS step, we convert a solution from unified representation to separated representation for each task. The RVNS then applies to each task separately. Finally, the output of the RVNS is converted into a unified representation for the next step. The improved solution will replace the original solution in the population.

For this step, we use several neighborhoods such as move, shift, swap-adjacent, exchange, 2-opt, and or-opt in (D. S. Johnson et al.,1995; C. R. Reeves et al.,1999). In addition, the pseudocode of the RVNS algorithm is given in Algorithm 7.

4.8 Elitism operator

Elitism is a process that ensures the survival of the fittest, so they do not die through the evolutionary processes. Researchers show the number (E. G. Talbi et al.,2009) (usually below 15%) of the best solutions that automatically go to the next generation. The proposed algorithm selects Sp individuals to the next generation, in which about 15% of them are the best solutions in the previous generation, and the remaining individuals are chosen randomly from P.

The stop condition: After the number of generations (Ng), the best solution has not been improved, the GA stops.

5 COMPUTATIONAL EVALUATIONS

The experiments are conducted on a personal computer equipped with a Xeon E-2234 CPU and 16 GB bytes RAM. The selection of parameters is implemented in preliminary experiments. The best configuration is presented as follows: finding the best configuration by conducting all instances would have been too expensive in computation, we test numerical analysis on some instances. The configuration selected in many combinations is tested, and the one that has obtained the best solution is chosen. In Table 1, we determine a range for each parameter that generates different combinations, and we run the proposed algorithm on some selected instances of the combinations. We find the following settings so that our algorithm obtains the best solutions: $SP = 100, NG = 5, rmp = 0.7, \alpha = 10, level = 5$, and $Ng = 100$. This parameter setting has thus been used in the following experiments.

We found no algorithm based on the literature's MFEA framework for the TRPTW and TSPTW. Therefore, the proposed algorithm's results directly compare with the known best solutions of the TSPTW and TRPTW on the same benchmark. Moreover, to compare with the previous MFEA framework (E. Osaba et al.,2020; Y. Yuan et al.,2016), our MEFA is tested on the benchmark for the TSP and TRP. They are specific variants of TSPTW and TRPTW without time window constraints. Therefore, the instances are used in the paper as follows:

- Dumas et al. proposes the first set citebib08 and contains 135 instances grouped in 27 test cases. Each group has five Euclidean instances, coordinates between 0 and 50, with the same number of customers and the same maximum range of time windows. For example, the instances n20w60.001, n20w60.002, n20w60.003, n20w60.004, and n20w60.005 have 20 vertices and the time window for each vertex is uniformly random, between 0 and 60.

- Gendreau et al. proposes the second set of instances citebib11 and contains 140 instances grouped in 28 test cases.

- Ohlmann et al. proposes the third set of instances citebib25 and contains 25 instances grouped in 5 test cases.
- The fourth sets in the majority are the instances proposed by Dumas et al. (Y. Dumas et al.,1995) with wider time windows.
- The TSPLIB [2] includes fourteen instances from 50 to 100 instances.

The efficiency of the metaheuristic algorithm can be evaluated by comparing the best solution found by the proposed algorithm (notation: *Best.Sol*) to 1) the optimal solution (notation: *OPT*); and 2) the known best solution (notation: *KBS*) of the previous metaheuristics (note that: In the TSPTW, *KBS* is the optimal solution) as follows:

$$gap[\%] = \frac{Best.Sol - KBS(OPT)}{KBS(OPT)} \times 100\% \quad (3)$$

The smaller and smaller the value of *gap* is, the better and better our solution is. All instances and found solutions are available in the link [3].

In Tables, *OPT*, *Aver.Sol* and *Best.Sol* are the optimal, average, and best solution after ten runs, respectively. Let *Time* be the running time such that the proposed algorithm reaches the best solution. Note that: Y. Yuan et al. supported the source code of their algorithm in (Y. Yuan et al.,2016) while the dMFEA-II algorithm (E. Osaba et al.,2020) was implemented again by us. Tables from 2 to 4 compare the proposed MFEA with the known best or optimal solutions for the TSPTW, and TRPTW instances (H.Abeledo et al.,2013; H.B. Ban et al.,2017; H.B. Ban et al.,2021; Y. Dumas et al.,1995; G. Heilporna et al.,2010; R. F. Silva et al.,2010; J. W. Ohlmann et al.,2007; G. Pesant et al.,1998; C. R. Reeves et al.,1999). Table 7 compares the MFEA with RVNS, OA (E. Osaba et al.,2020), and YA (Y. Yuan et al.,2016). In the TSP, the optimal solutions of the TSPLIB-instances are obtained by running Concord tool [4]. In the TRP, the optimal or best solutions are obtained from (H.Abeledo et al.,2013).

5.1 Comparisons with the TSPTW and TRPTW algorithms

The values of Table 5 are the average values of Tables from 2 to 4. The average difference with the optimal solution for the TSPTW is 0.56% even for the instances with up to 200 vertices. It shows that our solutions are near-optimal for the TSPTW. In addition, the proposed algorithm reaches the optimal solutions for the instances with up to 80 vertices for the TSPTW. For the TRPTW, our MFEA is better than the previous algorithms such as Ban et al. (H.B. Ban et al.,2017; H.B. Ban et al.,2021), and G. Heilporna (G. Heilporna et al.,2010) in the literature when the average *gap* is -0.28% (note that: Ban et al.'s and G. Heilporna's et al.'s algorithms is developed to solve the TRPTW only). The obtained results are impressive since it can be observed that the proposed algorithm finds not only near-optimal solutions but also the new best known ones for two problems simultaneously. It also indicates the efficiency of transferrable knowledge between optimization tasks in improving the solution quality.

It is unrealistic to expect that the proposed MFEA always gives better solutions than those of state-of-the-art metaheuristic algorithms for the TSPTW and TRPTW in all cases because their algorithms are designed to solve each problem independently. Table 6 shows that the efficient algorithm for the TSPTW may not be good for the TRPTW on the same instances and vice versa. On average, the optimal solution for the TSPTW using the TRPTW objective function is 9.7% worse than the optimal solution for the TRPTW. Similarly, the known best solution for the TRPTW using the TSPTW objective function is 20.6% worse than the optimal solution for the TSPTW. We conclude that if the proposed MFEA simultaneously reaches the good solutions that are close to the optimal solutions for both problems and even better than the state-of-the-art algorithms in many cases, we can say that the proposed MFEA with RVNS for multitasking is beneficial.

5.2 Comparison with the previous MFEA algorithms

Table 7 compares our results to those of two algorithms (E. Osaba et al.,2020; Y. Yuan et al.,2016) for some instances in both of two problems. The results show that the proposed algorithm receives better solutions than the others in many cases. The difference between our average result and the optimal value is about 2.59%. Obviously, our solution is near optimal one.

[2] http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/
[3] https://sites.google.com/soict.hust.edu.vn/mfea-tsptw-trptw/home
[4] https://www.math.uwaterloo.ca/tsp/concorde.html

Table 2. Comparison between our results with the best-found values for TSPTW and TRPTW instances proposed by Dumas et al. (Y. Dumas et al.,1995), and Silva et al. (R. F. Silva et al.,2010)

Instances	TSPTW	TRPTW	MFEA							
			TSPTW				TRPTW			
	OPT	*KBS*	*Best.Sol*	*Aver.Sol*	*gap*	*Time*	*Best.Sol*	*Aver.Sol*	*gap*	*Time*
n20w20.001	378	2528	378	378	0.0	3	2528	2528	0.0	3
n20w20.002	286	2560	286	286	0.0	2	2560	2560	0.0	2
n20w20.003	394	2671	394	394	0.0	2	2671	2671	0.0	2
n20w20.004	396	2975	396	396	0.0	6	2975	2975	0.0	6
n20w40.001	254	2270	254	254	0.0	2	2270	2270	0.0	2
n20w40.002	333	2679	333	333	0.0	5	2679	2679	0.0	5
n20w40.003	317	2774	317	317	0.0	3	2774	2774	0.0	2
n20w40.004	388	2568	388	388	0.0	2	2568	2568	0.0	3
n20w60.001	335	2421	335	335	0.0	3	2421	2421	0.0	2
n20w60.002	244	2176	244	244	0.0	2	2176	2176	0.0	2
n20w60.003	352	2694	352	352	0.0	2	2694	2694	0.0	2
n20w60.004	280	2020	280	280	0.0	2	2020	2020	0.0	2
n20w80.001	329	2990	329	329	0.0	2	2990	2990	0.0	3
n20w80.002	338	2669	340	340	0.6	1	2669	2669	0.0	2
n20w80.003	320	2643	320	320	0.0	2	2643	2643	0.0	2
n20w80.004	304	2627	306	306	0.7	3	2552	2552	-2.9	2
n20w100.001	237	2294	237	237	0.0	3	2269	2269	-1.1	3
n20w100.002	222	2082	222	222	0.0	2	2082	2082	0.0	2
n20w100.003	310	2416	310	310	0.0	2	2416	2416	0.0	3
n20w100.004	349	2914	349	349	0.0	1	2862	2862	-1.8	2
n40w20.001	500	7875	500	500	0.0	9	7875	7875	0.0	8
n40w20.002	552	7527	552	552	0.0	7	7527	7527	0.0	8
n40w20.003	478	7535	478	478	0.0	8	7535	7535	0.0	9
n40w20.004	404	7031	404	404	0.0	8	7031	7031	0.0	9
n40w40.001	465	7663	465	465	0.0	7	7663	7663	0.0	9
n40w40.002	461	7104	461	461	0.0	8	7104	7104	0.0	8
n40w40.003	474	7483	474	474	0.0	8	7483	7483	0.0	8
n40w40.004	452	6917	452	452	0.0	8	6917	6917	0.0	9
n40w60.001	494	7066	494	494	0.0	9	7066	7066	0.0	7
n40w60.002	470	7247	470	470	0.0	8	7247	7247	0.0	8
n40w60.003	408	6758	410	410	0.0	9	6758	6758	0.0	8
n40w60.004	382	5548	382	382	0.0	9	5548	5548	0.0	9
n40w80.001	395	8229	395	395	0.0	8	8152	8152	0.0	9
n40w80.002	431	7176	431	431	0.0	8	7123	7123	0.0	9
n40w80.003	412	7075	418	418	0.0	8	7075	7075	0.0	9
n40w80.004	417	7166	417	417	0.6	9	7166	7166	0.0	10
n40w100.001	429	6858	432	432	0.0	8	6800	6800	0.0	9
n40w100.002	358	6778	364	364	0.7	11	6693	6693	-2.9	10
n40w100.003	364	6178	364	364	0.0	9	6926	6926	-1.1	11
n40w100.004	357	7019	361	361	0.0	9	13996	13996	0.0	8
n60w20.002	605	13996	605	605	0.0	18	13782	13782	0.0	19
n60w20.003	533	13782	533	533	0.0	17	12965	12965	-1.8	18
n60w20.004	616	12965	616	616	0.0	17	15102	15102	0.0	18
n60w40.003	603	15034	612	612	0.0	19	15034	15034	0.0	19

Table 2. Comparison between our results with the best-found values for TSPTW and TRPTW instances proposed by Dumas et al. (Y. Dumas et al.,1995), and Silva et al. (R. F. Silva et al.,2010) (continue)

	TSPTW	TRPTW	MFEA							
			TSPTW				TRPTW			
Instances	*OPT/ KBS*	*KBS*	*Best.Sol*	*Aver.Sol*	*gap*	*Time*	*Best.Sol*	*Aver.Sol*	*gap*	*Time*
n60w160.004	401	11645	401	401	0.0	19	11778	11778	1.1	19
n60w180.002	399	12015	399	399	0.0	17	12224	12224	1.7	21
n60w180.003	445	12214	445	445	0.0	18	12679	12679	3.8	21
n60w180.004	456	11101	456	456	0.0	19	11245	11245	1.3	18
n60w200.002	414	11748	414	414	0.0	20	11866	11866	1.0	19
n60w200.003	455	10697	460	460	1.1	19	10697	10697	0.0	18
n60w200.004	431	11441	431	431	0.0	16	11740	11740	2.6	17
n80w120.002	577	18181	577	577	0.0	17	18383	18383	1.1	21
n80w120.003	540	17878	548	548	1.5	19	17937	17937	0.3	21
n80w120.004	501	17318	501	501	0.0	18	17578	17578	1.5	18
n80w140.002	472	17815	472	472	0.0	19	18208	18208	2.2	21
n80w140.003	580	17315	580	580	0.0	20	17358	17358	0.2	21
n80w140.004	424	18936	424	424	0.0	19	19374	19374	2.3	18
n80w160.002	553	17091	553	553	0.0	27	17200	17200	0.6	18
n80w160.003	521	16606	521	521	0.0	21	16521	16521	-0.5	20
n80w160.004	509	17804	509	509	0.0	20	17927	17927	0.7	18
n80w180.002	479	17339	479	479	0.0	21	17904	17904	3.3	19
n80w180.003	530	17271	530	530	0.0	20	17160	17160	-0.6	20
n80w180.004	479	16729	479	479	0.0	22	16849	16849	0.7	21
n100w120.002	540	29882	556	556	3.0	38	29818	29818	0.0	45
n100w120.003	617	25275	646	646	4.7	37	24473	24473	0.0	42
n100w120.004	663	30102	663	663	0.0	39	31554	31554	0.0	41
n100w140.002	622	30192	632	632	1.6	38	30087	30087	0.0	45
n100w140.003	481	28309	481	481	0.0	39	28791	28791	0.0	47
n100w140.004	533	27448	533	533	0.0	40	27990	27990	0.0	45
n150w120.003	747	42340	769	769	2.9	75	42339	42339	0.0	72
n150w140.001	762	42405	785	785	3.0	70	42388	42388	-0.1	74
n150w160.001	706	45366	732	732	3.6	72	45160	45160	-0.4	78
n150w160.002	711	44123	735	735	3.3	74	44123	44123	0.0	76
n200w200.001	9424	1094630	9424	9424	0.0	101	1093537	1093537	-0.1	89
n200w200.002	9838	1099839	9885	9885	0.5	110	1099839	1099839	0.0	86
n200w200.003	9043	1067171	9135	9135	1.0	99	1067161	1067161	0.0	93
n200w300.001	7656	1052884	7791	7791	1.7	100	1047893	1047893	-0.5	106
n200w300.002	7578	1047893	7721	7721	1.8	105	1047893	1047893	0.0	110
n200w300.003	8600	1069169	8739	8739	1.6	120	1069169	1069169	0.0	93
n200w300.004	8268	1090972	8415	8415	1.7	112	1090972	1090972	0.0	96
n200w300.005	8030	1022000	8190	8190	1.9	114	1016765	1016765	-5.1	98
n200w400.001	7420	1064456	7661	7661	3.2	109	1064456	1064456	0.0	100
aver					0.49	26.24			-0.11	25.6

Table 3. Comparison between our results with the best-found values for TSPTW and TRPTW instances proposed by Gendreau et al. (M. Gendreau et al.,1998), and Ohlmann et al. (J. W. Ohlmann et al.,2007)

Instances	TSPTW	TRPTW	MFEA							
			TSPTW				TRPTW			
	KBS	*KBS*	*Best.Sol*	*Aver.Sol*	*gap*	*T*	*Best.Sol*	*Aver.Sol*	*gap*	*Time*
n20w120.002	218	2193	218	218	0.0	2	2193	2193	0.0	3
n20w120.003	303	2337	303	303	0.0	4	2337	2337	0.0	2
n20w120.004	300	2686	300	300	0.0	2	2686	2686	0.0	2
n20w140.002	272	2330	272	272	0.0	2	2330	2330	0.0	3
n20w140.003	236	2194	236	236	0.0	2	2196	2196	0.1	2
n20w140.004	255	2279	264	264	3.5	4	2278	2278	0.0	5
n20w160.002	201	1830	201	201	0.0	2	1830	1830	0.0	2
n20w160.003	201	2286	201	201	0.0	3	2286	2286	0.0	2
n20w160.004	203	1616	203	203	0.0	2	1616	1616	0.0	2
n20w180.002	265	2315	265	265	0.0	4	2315	2315	0.0	2
n20w180.003	271	2414	271	271	0.0	2	2414	2414	0.0	2
n20w180.004	201	2624	201	201	0.0	3	1924	1924	-26.7	2
n20w200.002	203	1799	203	203	0.0	2	1799	1799	0.0	2
n20w200.003	249	2144	260	260	4.4	2	2089	2089	-2.6	1
n20w200.004	293	2624	293	293	0.0	1	2613	2613	-0.4	2
n40w120.002	445	6265	446	446	0.2	3	6265	6265	0.0	8
n40w120.003	357	6411	360	360	0.8	2	6411	6411	0.0	7
n40w120.004	303	5855	303	303	0.0	3	5855	5855	0.0	6
n40w140.002	383	5746	383	383	0.0	2	5746	5746	0.0	8
n40w140.003	398	6572	398	398	0.0	3	6572	6572	0.0	7
n40w140.004	342	5719	350	350	2.3	8	5680	5680	-0.7	8
n40w160.002	337	6368	338	338	0.3	9	6351	6351	-0.3	8
n40w160.003	346	5850	346	346	0.0	9	5850	5850	0.0	9
n40w160.004	288	4468	289	289	0.3	8	4440	4440	-0.6	9
n40w180.002	347	6104	349	349	0.6	8	6104	6104	0.0	9
n40w180.003	279	6040	282	282	1.1	7	6031	6031	-0.1	8
n40w180.004	354	6103	361	361	2.0	8	6283	6283	2.9	8
n40w200.002	303	6674	303	303	0.0	8	5830	5830	-12.6	9
n40w200.003	339	5542	343	343	1.2	7	5230	5230	-5.6	8
n40w200.004	301	6103	301	301	0.0	9	5977	5977	-2.1	10
n60w120.002	427	12517	427	427	0.0	19	12525	12525	0.1	19
n60w120.003	407	11690	419	419	2.9	19	11680	11680	-0.1	19
n60w120.004	490	11132	492	492	0.4	13	11135	11135	0.0	19
n60w140.002	462	11782	464	464	0.4	18	11810	11810	0.2	19
n60w140.003	427	13128	448	448	4.9	13	13031	13031	-0.7	16
n60w140.004	488	13189	488	488	0.0	15	12663	12663	-4.0	15
n60w160.002	423	12471	423	423	0.0	19	12719	12719	2.0	17
n60w160.003	434	10682	447	447	3.0	14	10674	10674	-0.1	15

Table 4. Comparison between our results with the best-found values for TSPTW and TRPTW instances proposed by Gendreau et al. (M. Gendreau et al.,1998), and Ohlmann et al. (J. W. Ohlmann et al.,2007) (continue)

Instances	TSPTW	TRPTW	MFEA							
			TSPTW				TRPTW			
	OPT	*KBS*	*Best.Sol*	*Aver.Sol*	*gap*	*Time*	*Best.Sol*	*Aver.Sol*	*gap*	*Time*
n60w160.004	401	11645	401	401	0.0	19	11778	11778	1.1	19
n60w180.002	399	12015	399	399	0.0	17	12224	12224	1.7	21
n60w180.003	445	12214	445	445	0.0	18	12214	12679	0.0	21
n60w180.004	456	11101	456	456	0.0	19	11245	11245	1.3	18
n60w200.002	414	11748	414	414	0.0	20	11866	11866	1.0	19
n60w200.003	455	10697	460	460	1.1	19	10697	10697	0.0	18
n60w200.004	431	11441	431	431	0.0	16	11441	11441	0.0	17
n80w120.002	577	18181	577	577	0.0	17	18383	18383	1.1	21
n80w120.003	540	17878	548	548	1.5	19	17937	17937	0.3	21
n80w120.004	501	17318	501	501	0.0	18	17578	17578	1.5	18
n80w140.002	472	17815	472	472	0.0	19	17815	17815	0.0	21
n80w140.003	580	17315	580	580	0.0	20	17358	17358	0.2	21
n80w140.004	424	18936	424	424	0.0	19	18936	18936	0.0	18
n80w160.002	553	17091	553	553	0.0	27	17200	17200	0.6	18
n80w160.003	521	16606	521	521	0.0	21	16521	16521	-0.5	20
n80w160.004	509	17804	509	509	0.0	20	17927	17927	0.7	18
n80w180.002	479	17339	479	479	0.0	21	17339	17339	0.0	19
n80w180.003	530	17271	530	530	0.0	20	17160	17160	-0.6	20
n80w180.004	479	16729	479	479	0.0	22	16849	16849	0.7	21
n100w120.002	540	29882	556	556	3.0	38	29818	29818	0.0	45
n100w120.003	617	25275	646	646	4.7	37	24473	24473	0.0	42
n100w120.004	663	30102	663	663	0.0	39	31554	31554	0.0	41
n100w140.002	622	30192	632	632	1.6	38	30087	30087	0.0	45
n100w140.003	481	28309	481	481	0.0	39	28791	28791	0.0	47
n100w140.004	533	27448	533	533	0.0	40	27990	27990	0.0	45
aver					0.64	13.6			-0.44	14.7

Figure 3. Comparing convergence trends of f_1 in multi-tasking and single-tasking for the TSPTW in n40w40 and n80w80 instances

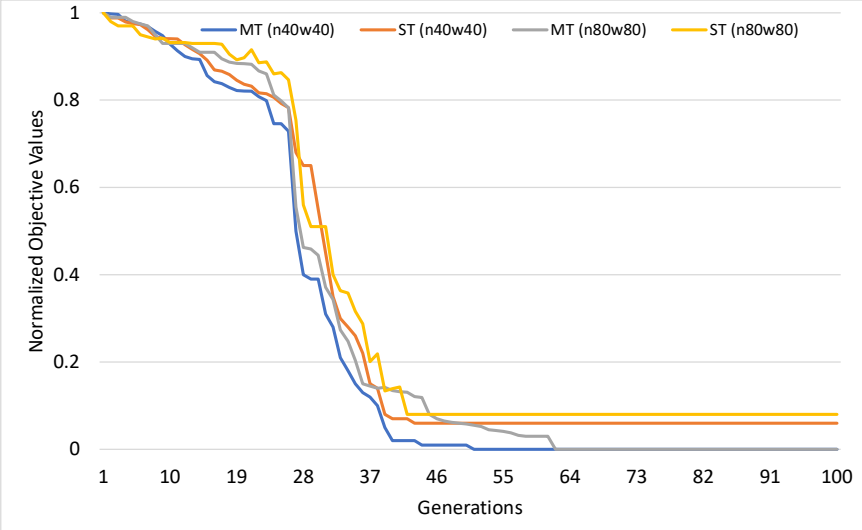


Table 5. Comparison between our results with the best-found values for TSPTW and TRPTW instances proposed by Gendreau et al. (M. Gendreau et al.,1998), and Ohlmann et al. (J. W. Ohlmann et al.,2007)

Instances	TSPTW	TRPTW		MFEA					
				TSP			TRP		
	OPT	*BKS*		*Best.Sol*	*Aver.Sol*	*Time*	*Best.Sol*	*Aver.Sol*	*Time*
		Ban et al.	Heilporna et al.						
n20w120.001	274	2175	2535	274	274	2	2175	2175	2
n20w140.001	176	1846	1908	176	176	2	1826	1826	2
n20w160.001	241	2146	2150	241	241	2	2148	2148	2
n20w180.001	253	2477	2037	253	253	2	2477	2477	2
n20w200.001	233	1975	2294	233	233	2	1975	1975	2
n40w120.001	434	6800	7496	434	434	9	6800	6800	9
n40w140.001	328	6290	7203	328	328	10	6290	6290	10
n40w160.001	349	6143	6657	349	349	11	6143	6143	12
n40w180.001	345	6952	6578	345	345	12	6897	6897	11
n40w200.001	345	6169	6408	345	345	10	6113	6113	13
n60w120.001	392	11120	9303	392	392	25	11288	11288	28
n60w140.001	426	10814	9131	426	426	26	10981	10981	27
n60w160.001	589	11574	11422	589	589	27	11546	11546	28
n60w180.001	436	11363	9689	436	436	24	11646	11646	25
n60w200.001	423	10128	10315	423	423	25	9939	9939	27
n80w120.001	509	11122	11156	512	509	41	16693	16693	45
n80w140.001	530	14131	14131	530	530	42	14131	14131	47
n80w180.001	605	11222	11222	605	605	41	11222	11222	42

Table 6. The average results for TSPTW, TRPTW instances

Instances	TSPTW		TRPTW	
	$\overline{gap}$	*Time*	$\overline{gap}$	*Time*
TSPTW	0.49	26.24	-0.11	26.5
TSPTW	0.64	13.6	-0.44	14.7
aver	0.56	19.9	-0.28	20.6

Table 7. The difference between the optimal TSPTW using TRPTW objective function and vice versa

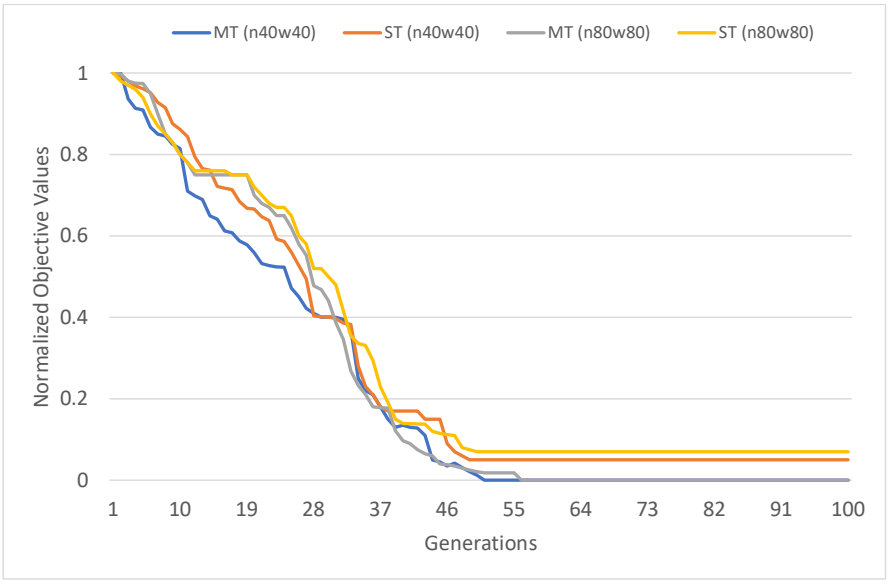
Instances	TRPTW			TSPTW		
	TRPTW(OPT_TSPTW)	KBS	diff[%]	TSPTW(BKS_TRPTW)	KBS	diff[%]
n20w120.002	2592	2193	15.4	256	218	14.8
n20w140.002	2519	2330	7.5	311	272	12.5
n20w160.002	2043	1830	10.4	249	201	19.3
n40w120.002	6718	6265	6.7	552	446	19.2
n40w140.002	5865	5746	2.0	449	383	14.7
n40w160.002	7519	6351	15.5	456	338	25.9
n60w120.002	13896	12517	9.9	581	444	23.6
n60w140.002	12898	11795	8.6	616	464	24.7
n60w160.002	14091	12489	11.4	616	428	30.5
aver			9.7			20.6

Table 8. Comparison between our results with the others for TSP and TRP in TSPLIB (**?**)

Instances	OPT		YA (**?**)		OA (J. N. Tsitsiklis et al.,1992)		MFEA							
	TSP	TRP	TSP	TRP	TSP	TRP	TSP				TRP			
			best.sol	*best.sol*	*best.sol*	*best.sol*	*best.sol*	*aver.sol*	*gap*	*Time*	*best.sol*	*aver.sol*	*gap*	*Time*
eil51	426*	10178*	446	10834	450	10834	426	426	0.00	23	10178	10178	0.00	22
berlin52	7542*	143721*	7922	152886	8276	152886	7542	7542	0.00	22	143721	143721	0.00	21
st70	675*	20557*	713	22283	772	22799	680	680	0.01	41	22283	22283	8.40	39
eil76	538*	17976*	560	18777	589	18008	559	559	0.04	43	18008	18008	0.18	40
pr76	108159*	3455242*	113017	3493048	117287	3493048	108159	108159	0.00	47	3455242	3455242	0.00	45
pr107	44303*	2026626*	45737	2135492	46338	2135492	45187	45187	0.02	71	2052224	2052224	1.26	72
rat99	1211*	58288*	1316	60134	1369	60134	1280	1280	0.06	66	58971	58971	1.17	65
kroA100	21282*	983128*	22233	1043868	22233	1043868	21878	21878	0.03	63	1009986	1009986	2.73	63
kroB100	22141*	986008*	23144	1118869	24337	1118869	23039	23039	0.04	64	1003107	1003107	1.73	63
kroC100	20749*	961324*	22395	1026908	23251	1026908	21541	21541	0.04	68	1007154	1007154	4.77	66
kroD100	21294*	976965*	22467	1069309	23833	1069309	22430	22430	0.05	70	1019821	1019821	4.39	72
kroE100	22068*	971266*	22960	1056228	23622	1056228	22964	22964	0.04	60	1034760	1034760	6.54	64
rd100	7910*	340047*	8381	380310	8778	365805	8333	8333	0.05	63	354762	354762	4.33	64
eil101	629*	27519*	681	28398	695	28398	662	662	0.05	62	27741	27741	0.81	61
aver									0.03				2.59	

'*' is the optimal values

Figure 4. Comparing convergence trends of f_2 in multi-tasking and single-tasking for the TRPTW in n40w40 and n80w80 instances



5.3 Convergence trend

The normalized objective function is used for analyzing the proposed MFEA algorithm's convergence trends. It calculated as follows:

$$\overline{f_j} = \frac{(f_j - f_j^{min})}{(f_j^{max} - f_j^{min})},$$

where $j = 1, 2$ and f_j^{min}, f_j^{max} are the minimum and maximum function cost values across all test runs.

The convergence trend of the two strategies is described in Figures 3 and 4 for n40w40 and n80w80 instances. Single-tasking (ST) converges faster than multitasking (MT) in these figures. That means multitasking can generally converge to a better objective value.

In summary, the performance of the multitasking strategy is better than the one of single-tasking. With the same instances and algorithm, exploiting multiple function landscapes via implicit genetic transfer can lead the improvement. The process of transferring knowledge during multitasking leads to the clear dominance of multitasking in comparison with single-tasking. It is the impressive advantage of the evolutionary multitasking paradigm.

The proposed algorithm's average running time does not consume in Tables than the others in (E. Osaba et al.,2020; Y. Yuan et al.,2016). Moreover, it grows quite moderate with single-tasking because it runs two problems simultaneously.

6 DISCUSSIONS AND CONCLUSIONS

The MFEA framework (K.K. Bali et al.,2019; A. Gupta et al.,2016; E. Osaba et al.,2020; Y. Yuan et al.,2016; Q. Xu et has been proposed to incorporate Evolutionary into multitasking to handle multiple problems at the same time. Instead of solving a pool of similar optimization problems independently, it performs multiple tasks for systems. Therefore, it can be useful in a system with limited computation. In addition, the advantage of the approach in comparison with single-task EA is that the phenomenon of genetic information transfer in multitasking can exploit transferrable knowledge between optimization tasks. Therefore, it can find better solutions for multitasks. That is an important characteristic of the MEFA.

The TSPTW, and TRPTW are combinatorial optimization problems that have many practical situations. Currently, there exist many algorithms that are proposed to solve them. We can find the TSPTW's and TRPTW's algorithms in the literature and run them in parallel. However, these algorithms are designed to solve each problem independently and separately. They do not use a unified search space. Therefore, they cannot exploit positive transferrable knowledge between optimization tasks. In addition, running two algorithms in parallel can require a strong enough computation system while our MFEA runs sequentially. The MFEA is suitable in a system with limited computation. This paper introduces the first algorithm that combines the MFEA framework and RVNS for solving two tasks simultaneously. The combination is to have positive transferrable knowledge between tasks from the MFEA and the ability to exploit good solution spaces from RVNS. Due to the important characteristics, finding better solutions will be increased. In the literature, the idea of the combination between the MFEA and local search (such as 2-opt) is proposed in (Q. Xu et al.,2021). However, 2-opt may not exploit well promising solution space. Conversely, the RVNS is a powerful framework that uses many neighborhood search heuristics. It often exploits promising solution space better than 2-opt. In comparison with the previous schemes, our scheme includes new features as follows:

- In the initial variant of MFEA, all tasks are shared the genetic information without considering its impact on solving the tasks. Therefore, genetic information transfer can be negative or positive. We propose a new selection operator that balances skill-factor and population diversity. The skill-factor is effectively transferred elite genes between tasks, while diversity in the population is important when it meets a bottleneck against the information transfer.
- Multiple crossover and mutation operator schemes are used in our MFEA. These schemes help our algorithm to maintain good diversity.
- The combination between the MFEA with the RVNS to have positive transferrable knowledge between tasks from the MFEA and the ability to exploit good solution spaces from RVNS. Our RVNS uses six neighborhoods; therefore, the explored neighborhood is extended, and the chance to obtain a better solution is high.

Summarily, this work's main contributions can be summarized as follows: 1) from the algorithmic perspective, the proposed algorithm brings the advantages of the MFEA with multiple crossover and mutation operators and RVNS. The hybrid consists of new features compared with the previous schemes; 2) from the computational perspective, extensive numerical experiments on benchmark instances show that our algorithm solves two problems well simultaneously. Moreover, it reaches better solutions than the previous MFEA framework in many cases. More interestingly, it finds the new best-known solutions compared to the state-of-the-art metaheuristics only for the TRPTW in many cases. It indicates the efficiency of transferrable knowledge between optimization tasks in the MFEA framework.

REFERENCES

[H.Abeledo et al.,2013] H. Abeledo, R. Fukasawa, A. Pessoa, and E. Uchoa, The time dependent traveling salesman problem: polyhedra and algorithm, J. Mathematical Programming Computation, 5, 2013, pp. 27-55.

[H.B. Ban et al.,2017] H.B. Ban, N.D. Nghia, Metaheuristic for the Traveling Repairman Problem with Time Windows, Proc. RIVF, 2017, pp.1-6.

[H.B. Ban et al.,2021] H.B. Ban, A metaheuristic for the delivery man problem with time windows, J. Joco, 41 (4), 2021, pp. 794-816.

[K.K. Bali et al.,2019] K.K. Bali, Y.S. Ong, A. Gupta, P.S. Tan, Multifactorial evolutionary algorithm with online transfer parameter estimation: Mfea-II, J. IEEE Trans Evol Comput, 24(1), 2019, pp. 69–83.

[W.B. Carlton et al.,1996] W.B. Carlton, J.W. Barnes, Solving the travelling salesman problem with time windows using tabu search, J. IEE Transactions, 28, 1996, pp. 617-629.

[S. Dash et al.,2012] S. Dash, O. Günlük, A. Lodi, and A. Tramontani. A Time Bucket Formulation for the Traveling Salesman Problem with Time Windows, INFORMS Journal on Computing, 24, pp 132-147, 2012.

[M. Dorigo et al.,1997] M. Dorigo, L.M. Gambardella, Ant colony system: a cooperative learning approach to the traveling salesman problem, J. IEEE Trans. Evol. Comput, 1(1), 1997, pp. 53–66.

[Y. Dumas et al.,1995] Y. Dumas, J. Desrosiers, E. Gélinas, An optimal algorithm for the Traveling Salesman Problem with Time Windows, J. Operations Research, 43, 1995, pp. 367-371.

[T. A. Feo et al.,1995] T. A. Feo, and M.G.C. Resende, Greedy Randomized Adaptive Search Procedures, J. Global Opt., 1995, pp. 109-133.

[F. Focacci et al.,2002] F. Focacci, A. Lodi, M. Milano, A hybrid exact algorithm for the TSPTW, INFORMS Journal on Computing, 14 (4), 2002, pp. 403-417.

[M. Gendreau et al.,1998] M. Gendreau, A. Hertz, G. Laporte, M. Stan, A generalized insertion heuristic for the traveling salesman problem with time windows, J. Operations Research, 43, 1998, pp. 330–335.

[A. Gupta et al.,2016] A. Gupta, Y.S. Ong, L. Feng, Multifactorial evolution: toward evolutionary multitasking, J. IEEE Trans Evol Comput, 20(3), pp. 343–357, 2016.

[G. Heilporna et al.,2010] G. Heilporna, Jean-François Cordeaua, and Gilbert Laporte, The Delivery Man Problem with time windows, 7, 2010, pp. 269-282.

[T. Ibaraki et al.,2008] T. Ibaraki, S. Imahori, K. Nonobe, K. Sobue, T. Uno, and M. Yagiura, An iterated local search algorithm for the vehicle routing problem with convex time penalty functions, J. Discrete Applied Mathematics, 11 (156), 2008, pp. 2050–2069.

[A. Langevin et al.,1993] A. Langevin, M. Desrochers, J. Desrosiers, Sylvie Gélinas, and F. Soumis. A two-commodity flow formulation for the traveling salesman and makespan problems with time windows. Networks, 23(7), 1993, pp. 631-640.

[Y. C. Lian et al.,2019] Y. C. Lian, Z. X. Huang, Y. R. Zhou, Z. F. Chen, Improve theoretical upper bound of Jumpk function by evolutionary multitasking, Proc. HPCCT, pp. 22–24, 2019, pp. 44–50.

[O. Martin et al.,1991] O. Martin, S. W. Otto, E. W. Felten, Large-step Markov Chains for the Traveling Salesman Problem, J. Complex Systems, 5 (3), 1991, pp. 299–326.

[E. Osaba et al.,2020] E. Osaba, A.D. Martinez, A. Galvez, A. Iglesias, J. Del Ser, dMFEA-II: An Adaptive Multifactorial Evolutionary Algorithm for Permutation-based Discrete Optimization Problems, Proc. GECCO, pp. 1690–1696, 2020.

[A. Otman et al.,2015] A. Otman, and A. Jaafar, A Comparative Study of Adaptive Crossove for Genetic Algorithms to Resolve the Traveling Salesman Problem, J. Computer Applications, 31 (11), pp. 49-57, 2011.

[N. Mladenovic et al.,1997] N. Mladenovic, P. Hansen, Variable Neighborhood Search, J. Operations Research, 24 (11), 1997, pp.1097-1100.

[R. F. Silva et al.,2010] R. F. Silva, S. Urrutia, A General VNS heuristic for the traveling salesman problem with time windows, J. Discrete Optimization, 7 (4), 2010, pp. 203-211.

[D. S. Johnson et al.,1995] D. S. Johnson, and L. A. McGeoch, The traveling salesman problem: A case study in local optimization in Local Search in Combinatorial Optimization, E. Aarts and J. K. Lenstra, eds., 1995, pp. 215-310.

[J. N. Tsitsiklis et al.,1992] J. N. Tsitsiklis, Special cases of Traveling Salesman and Repairman Problems with time windows, J. Networks, 22, 1992, pp. 263–283.

[E. G. Talbi et al.,2009] E. G. Talbi, Metaheuristics: from design to implementation, NewJersey, Wiley, 2009.

[J. W. Ohlmann et al.,2007] J. W. Ohlmann and B. W. Thomas, A Compressed-annealing Heuristic for the Traveling Salesman Problem with time windows, J. INFORMS, 19 (1), pp. 80-90, 2007.

[G. Pesant et al.,1998] G. Pesant, M. Gendreau, J.-Y. Potvin, and J.-M. Rousseau. An exact constraint logic programming algorithm for the traveling salesman problem with time windows. Transportation Science, 32:12-29, 1998.

[M. W. Salvesbergh et al.,1985] M. W. Salvesbergh, Local search in routing problems with time windows, J. Annals of Operations Research, 4, 1985, pp. 285-305

[C. R. Reeves et al.,1999] C. R. Reeves, Landscapes, operators and heuristic search, Annals of Operations Research 86(0), 1999, pp. 473-490.

[R. Roberti et al.,2016] R. Roberti, M. Wen, The Electric Traveling Salesman Problem with Time Windows, Transportation Research Part E: Logistics and Transportation Review, 89, 2016, pp. 32-52.

[Y. Yuan et al.,2016] Y. Yuan, Y.S. Ong, A. Gupta, P.S. Tan, H. Xu, Evolutionary multitasking in permutation-based combinatorial optimization problems: realization with tsp, qap, lop, and jsp, Proc. TENCON, 2016, pp. 3157-3164.

[Q. Xu et al.,2021] Q. Xu, N. Wang, L. Wang, W. Li, and Q. Sun, “Multi-Task Optimization and Multi-Task Evolutionary Computation in the Past Five Years: A Brief Review, J. Mathematics, 9 (864), 2021, pp. 1-44.